Математическая статистика

Практическое задание 3

В данном задании рассматриваются свойства условного математического ожидания. В частности, рассматривается модель смеси гауссовских распределений.

Правила:

- Выполненную работу нужно отправить на почту `probability.diht@yandex.ru`, указав тему письма `"[номер группы] Фамилия Имя - Задание 3"`. Квадратные скобки обязательны. Вместо `Фамилия Имя` нужно подставить свои фамилию и имя.
- Прислать нужно ноутбук и его pdf-версию. Названия файлов должны быть такими: `3.N.ipynb` и `3.N.pdf`, где `N` - ваш номер из таблицы с оценками.
- Никакой код из данного задания при проверке запускаться не будет.
- Некоторые задачи отмечены символом *. Эти задачи являются дополнительными. Успешное выполнение большей части таких задач (за все задания) является необходимым условием получения бонусного балла за практическую часть курса.
- Баллы за каждую задачу указаны далее. Если сумма баллов за задание меньше 25% (без учета доп. задач), то все задание оценивается в 0 баллов.

Баллы за задание:

- Задача 1 - 3 балла
- Задача 2 - 1 балл
- Задача 3 - 2 балла
- Задача 4 - 7 баллов
- Задача 5* - 10 баллов

Задача 1. На вероятностном пространстве $(\mathbb{R}_+, \mathcal{B}(\mathbb{R}_+), \mathsf{P})$, где P --- экспоненциальное распределение с параметром λ, задана случайная величина ξ по правилу $\xi(\omega) = \omega$. Сигма-алгебра $\mathcal{G}$ порождена счетной системой событий $\{B_n\}_{n\geq 1}$, где $B_n = \{n-1 \leq \omega < n\}$.. Для $\omega \in [0, 5]$ постройте графики

- плотности распределения P для $\lambda \in \{1, 3, 10\}$
- ξ и $\mathsf{E}(\xi|\mathcal{G})$ как функции от ω для $\lambda \in \{1, 3, 10\}$
- ξ^2 и $\mathsf{E}(\xi^2|\mathcal{G})$ как функции от ω для $\lambda \in \{1, 3, 10\}$

Используйте приведенный ниже шаблон. Одному и тому же значению λ во всех графиках должен соответствовать один и тот же цвет.

In [1]:

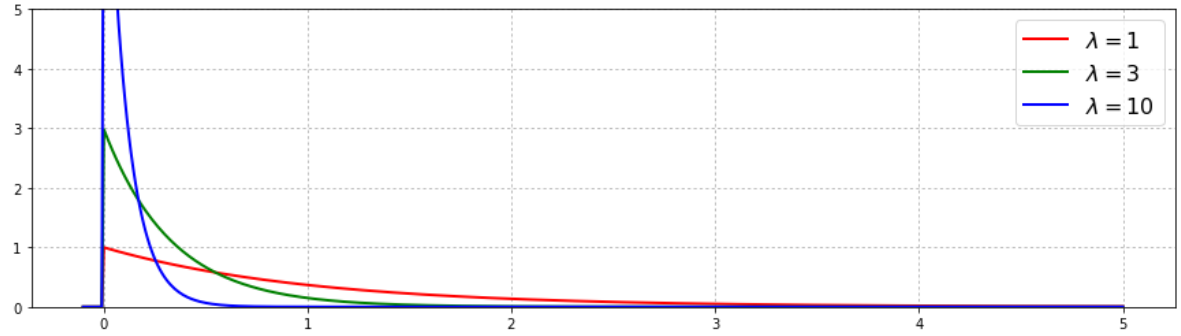
```
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as sps
%matplotlib inline
```

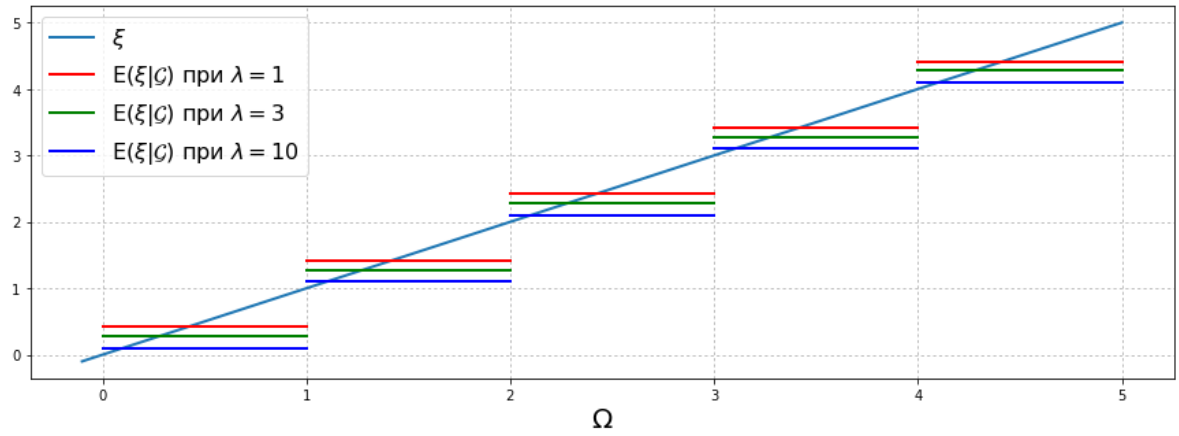
In [2]:

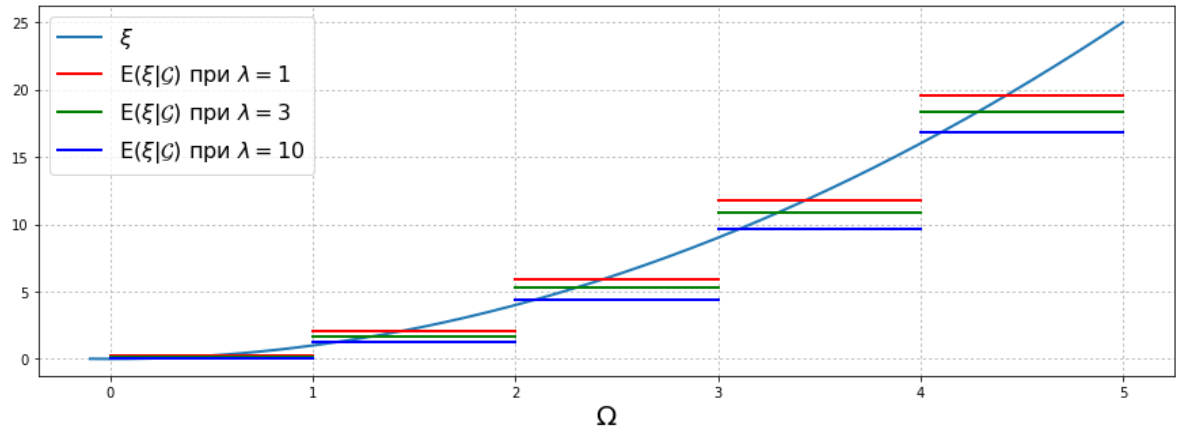
```
import math
```

In [4]:

```
lambdas = [(1,'r'), (3,'g'), (10, 'b')]
sample = np.linspace(-0.1, 5, 500)

# График 1
plt.figure(figsize=(15, 4))
for lambda_ in lambdas:
    plt.plot(sample, sps.expon(scale=1/lambda_[0]).pdf(sample), lw=2, color=lambda_[1],
             label='$\\lambda={}$'.format(lambda_[0]))
plt.legend(fontsize=16)
plt.ylim((0, 5))
plt.grid(ls=':')

# График 2
plt.figure(figsize=(15, 5))
plt.plot(sample, sample, lw=2, label='$\\xi$')
def f1(x, l):
    return -(math.e ** (-l * x)) * (x + 1 / l)
def f2(x, l):
    return -(math.e ** (-l * x))

for lambda_ in lambdas:
    for i in np.arange(1, 6, 1):  # события из сигма-алгебры
        y = (f1(i, lambda_[0])-f1(i-1, lambda_[0])) / (f2(i, lambda_[0]) - f2(i-1, lambda_[
        plt.plot((i-1, i), (y, y), color=lambda_[1], lw=2,
                 label=('$\\mathsf{E}(\\xi|\\mathcal{G})$ при $\\lambda = ' + str(lambda_
                        + '$') if i == 1 else '')
plt.xlabel('$\\Omega$', fontsize=20)
plt.legend(fontsize=16)
plt.grid(ls=':')

# График 3 для \xi^2 аналогичен графику 2
plt.figure(figsize=(15, 5))
plt.plot(sample, sample ** 2, lw=2, label='$\\xi$')
def f1(x, l):
    return -(math.e ** (-l * x)) * (x**2 + 2 * x / l + 2 / (l**2))
def f2(x, l):
    return -(math.e ** (-l * x))

for lambda_ in lambdas:
    for i in np.arange(1, 6, 1):  # события из сигма-алгебры
        y = (f1(i, lambda_[0])-f1(i-1, lambda_[0])) / (f2(i, lambda_[0]) - f2(i-1, lambda_[
        plt.plot((i-1, i), (y, y), color=lambda_[1], lw=2,
                 label=('$\\mathsf{E}(\\xi|\\mathcal{G})$ при $\\lambda = ' + str(lambda_
                        + '$') if i == 1 else '')
plt.xlabel('$\\Omega$', fontsize=20)
plt.legend(fontsize=16)
plt.grid(ls=':')
```







Вывод: Видим, что условное матожидание усредняет значение ξ на каждом B_n, что и должно быть:)

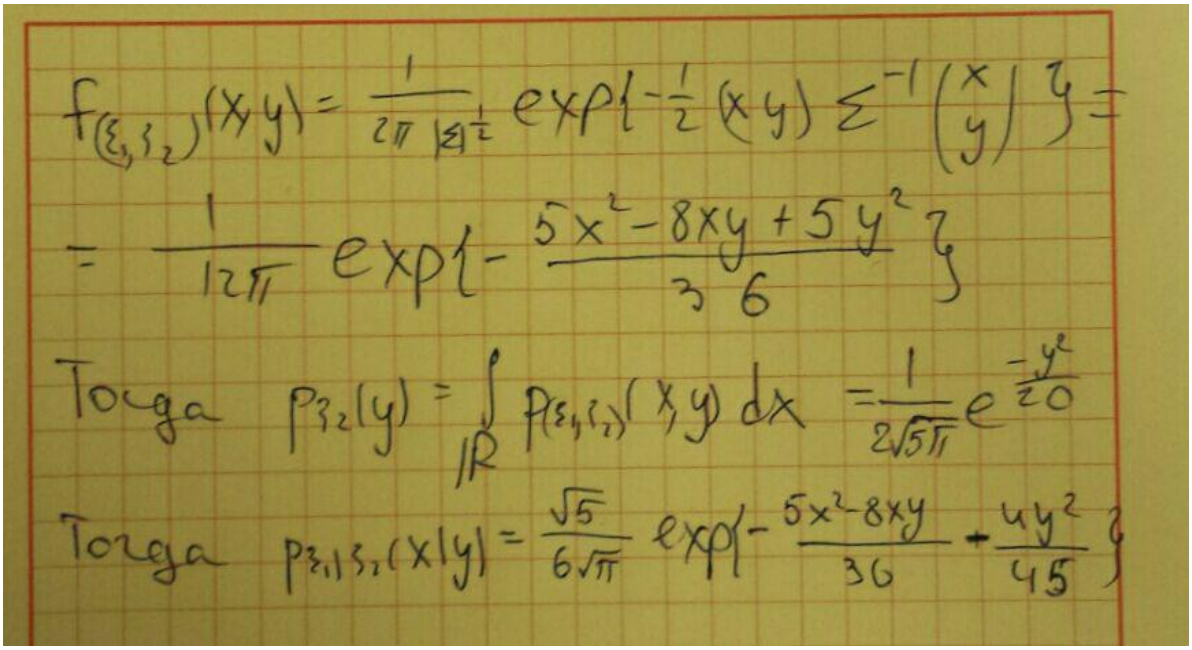
Задача 2. Пусть $\xi = (\xi_1, \xi_2) \sim \mathcal{N}(a, \Sigma)$, где $a = 0$ и $\Sigma = \begin{pmatrix} 10 & 8 \\ 8 & 10 \end{pmatrix}$. Для $y \in \{-3, 0, 1, 5\}$ постройте графики условной плотности $f_{\xi_1|\xi_2}(x\,|y)$.

Мне было лень это набирать, так что как-то так:)

In [5]:

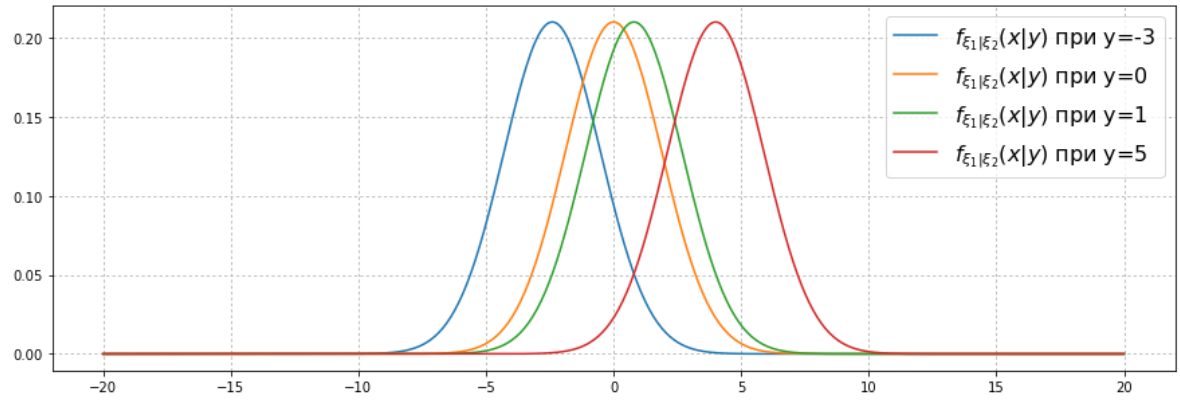
```
from IPython.display import Image
Image(filename='task2.jpg')
```

Out[5]:



In [6]:

```
y_ = [-3, 0, 1, 5]
sample = np.linspace(-20, 20, 500)
def f(x, y):
    return np.sqrt(5 / np.pi) / 6 * np.exp(-5 * x**2 / 36 + 2 / 9 * x * y - 4 * y**2 / 45)
plt.figure(figsize=(15, 5))
for y in y_:
    plt.plot(sample, f(sample, y), label='$f_{\\xi_1|\\xi_2} \\left(x|y\\right)$ при y=' +
plt.legend(fontsize=16)
plt.grid(ls=":")
plt.show()
```



Вывод

Задача 3. Имеется множество серверов, которые периодически выходят из строя. Обозначим ξ_i время между i-м моментом выхода из строя сервера и $(i+1)$-м. Известно, что величины ξ_i независимы в совокупности и имеют экспоненциальное распределение с параметром λ.

Обозначим N_t --- количество серверов, которые вышли из строя к моменту времени t (в начальный момент времени $N_0 = 0$). В курсе случайных процессов будет доказано, что для любых $s < t$ величина $N_t - N_s \sim Pois(\lambda(t-s))$ и независима с N_s. При этом N_t как функция от t будет называться пуассоновским процессом интенсивности λ.

Вам нужно знать, сколько серверов нужно докупить к моменту времени t взамен вышедших из строя. В момент времени s предсказанием количества серверов, вышедших из строя к моменту времени t, будем считать величину $\mathsf{E}(N_t|N_s)$.

Сгенерируйте выборку случайных величин ξ_i для $\lambda = 1/4$ в количестве, чтобы их сумма была больше 100. Для $t = 100$ постройте графики зависимости величины $\mathsf{E}(N_t|N_s)$ от s в предоположении, что условное математическое ожидание было посчитано при значении $\lambda \in \{1/10, 1/4, 1/2, 1\}$. Нарисуйте также на графике горизонтальную прямую уровня N_{100}.

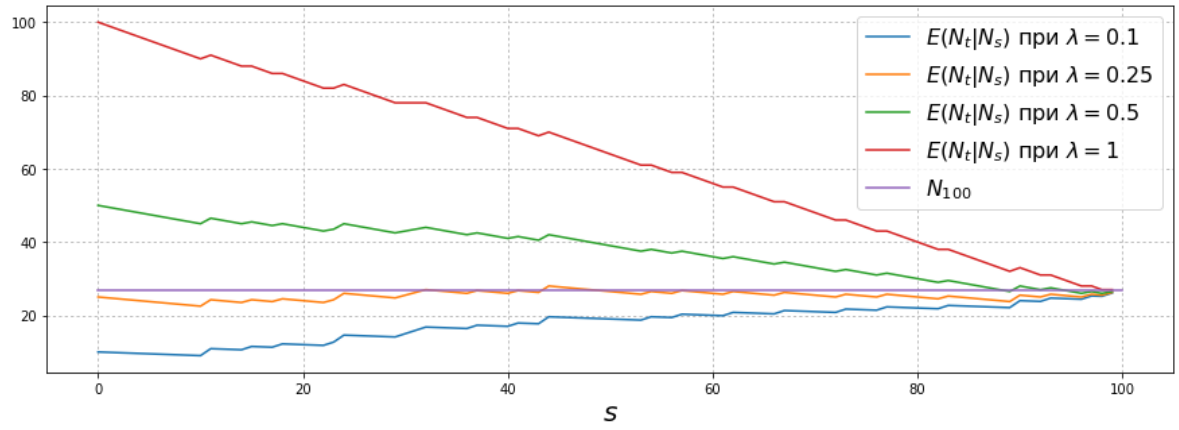
In [10]:

```
l = 1/4
sample = sps.expon(scale=1/l).rvs(100)
rs = 0
for val in sample:
    rs += val
print(rs > 100)
```

```
True
```

In [11]:

```
#считаем N_s
tmp = sample.cumsum()
N = [0]
for i in range(len(tmp)):
    while(len(N) - 1 < tmp[i]):
        N.append(i)
#E(N_t|N_s) = lambda(t-s) + N_s
lambdas = [1/10, 1/4, 1/2, 1]
plt.figure(figsize=(15, 5))
for lamb in lambdas:
    E = N.copy()
    for i in range(0, 100):
        E[i] += lamb * (100 - i)
    x = np.arange(0, 100, 1)
    plt.plot(x, E[:100], label='$E( N_t | N_s)$ при $\lambda={}$'.format(lamb))
plt.plot((0, 100), (N[100], N[100]), label='$N_{100}$')
plt.legend(fontsize=16)
plt.xlabel('$s$', fontsize=20)
plt.grid(ls=":")
plt.show()
```



Вывод:

Видим, что N_{100} лучше всего приближает предсказание $\mathrm{E}(N_t|N_s)$ при $\lambda = 0.25$, что и ожидалось, ведь мы строили выборку именно для такого λ. Т.е. по графику мы предсказали, что время выхода из строя ~ $Expon(0.25)$

Задача 4. Рассмотрим модель смеси многомерных гауссовских распределений, то есть распределение, имеющее плотность $p(x) = \sum_{k=1}^{K} p_k(x)\mathrm{P}(T = k)$, где T --- случайная величина, принимающая значения $\{1, \ldots, K\}$ и имеющая смысл номера компоненты смеси, а $p_k(x)$ --- плотность распределения $N(a_k, \Sigma_k)$.

Загрузите датасет "Ирисы Фишера", используя следующий код.

In [88]:

```
from sklearn.datasets import load_iris
data = load_iris()
sample = data['data']  # выборка
classes = data['target']  # номера компонент смеси
```

В предположении, что каждый класс имеет гауссовское распределение, оцените его параметры. Используйте для этого функции `numpy.mean` и `numpy.cov`. Проверьте, что матрица ковариаций получилась правильной --- возможно, придется предварительно поменять порядок осей (транспонировать). Напечатайте полученные оценки.

In [89]:

```
samples = [sample[:50], sample[50:100], sample[100:150]]
means = []
covs = []
```

In [90]:

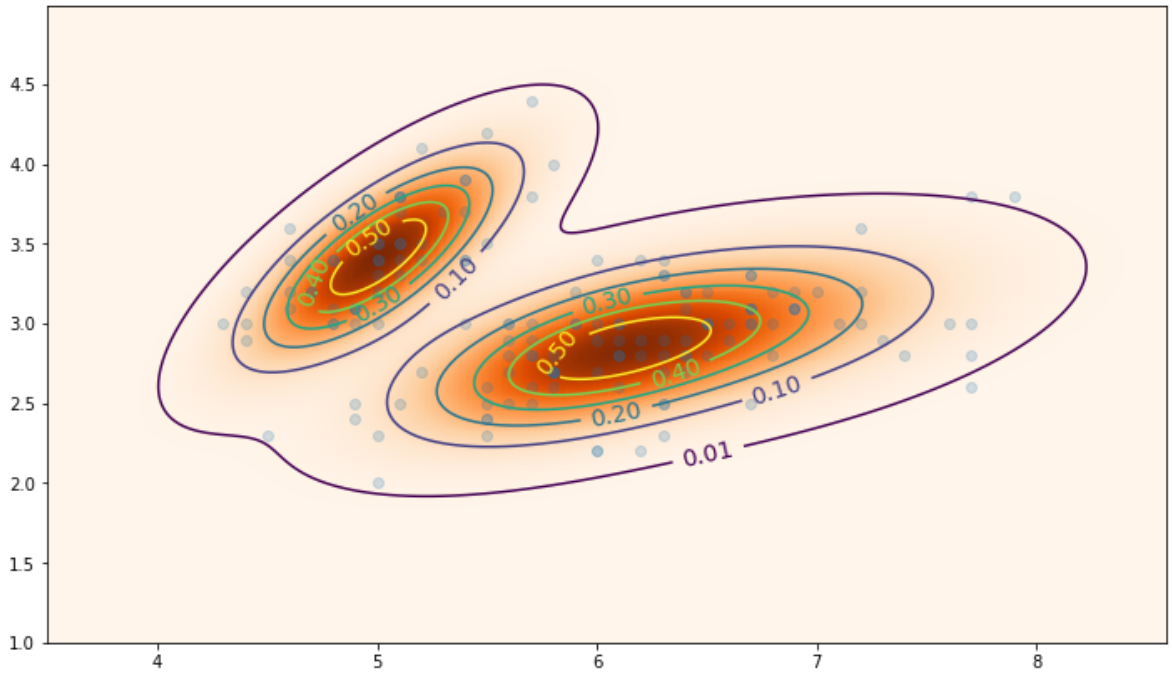
```
for sample_ in samples:
    means.append(np.mean(sample_, axis=0))
    covs.append(np.cov(sample_.T, bias=True))
for i in range(0, 3):
    print('mean', means[i])
    print('cov', covs[i])
```

```
mean [ 5.006  3.418  1.464  0.244]
cov [[ 0.121764  0.098292  0.015816  0.010336]
 [ 0.098292  0.142276  0.011448  0.011208]
 [ 0.015816  0.011448  0.029504  0.005584]
 [ 0.010336  0.011208  0.005584  0.011264]]
mean [ 5.936  2.77   4.26   1.326]
cov [[ 0.261104  0.08348   0.17924   0.054664]
 [ 0.08348   0.0965    0.081     0.04038 ]
 [ 0.17924   0.081     0.2164    0.07164 ]
 [ 0.054664  0.04038   0.07164   0.038324]]
mean [ 6.588  2.974  5.552  2.026]
cov [[ 0.396256  0.091888  0.297224  0.048112]
 [ 0.091888  0.101924  0.069952  0.046676]
 [ 0.297224  0.069952  0.298496  0.047848]
 [ 0.048112  0.046676  0.047848  0.073924]]
```

Нарисуйте график плотности (тепловую карту) в проекции на первые две координаты и нанесите на график точки выборки. При выполнении задания полезно вспомнить решение части 3 задачи 1 задания 1. Используйте шаблон ниже.

In [91]:

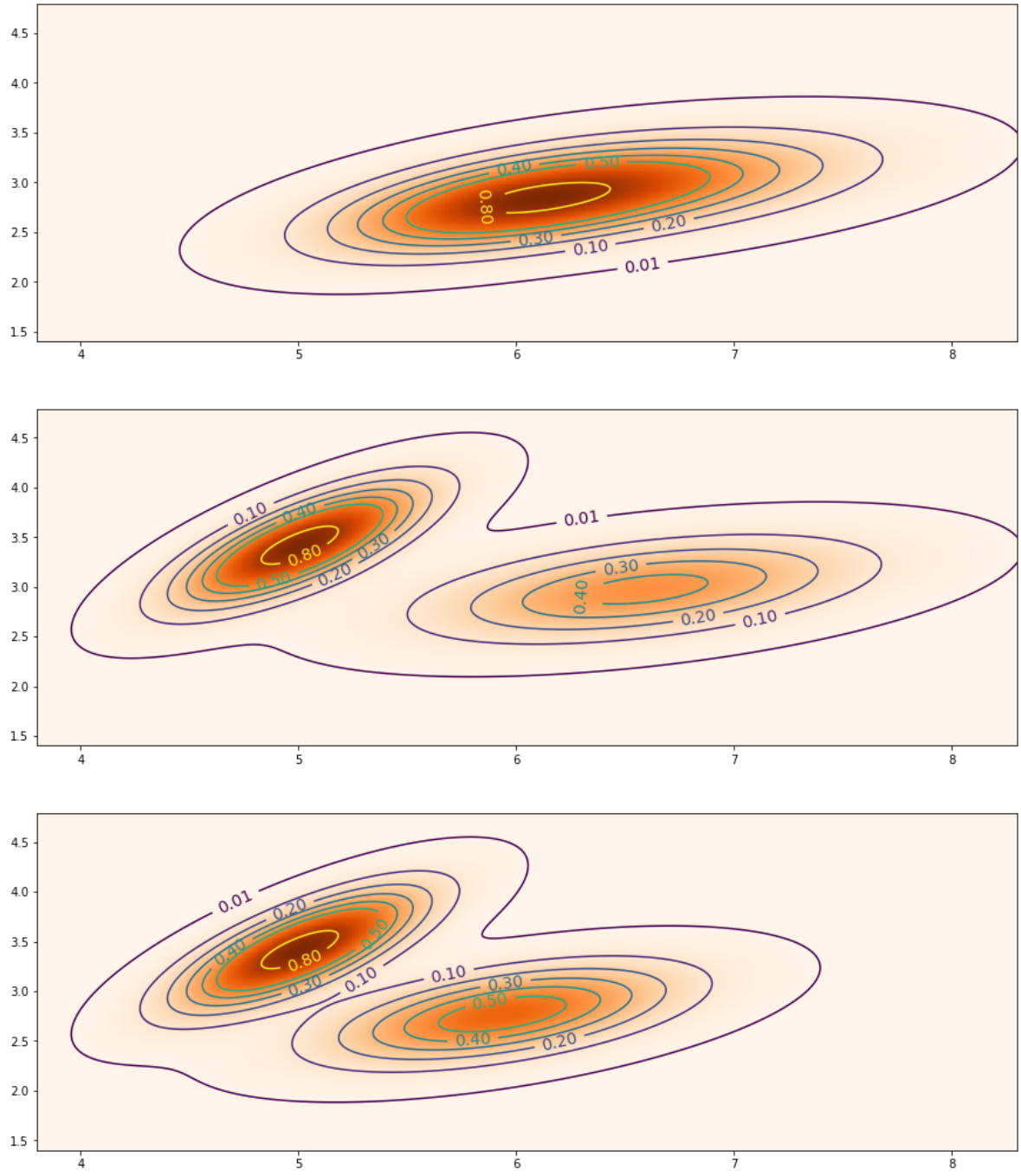
```
#копипаста из 3 части задачи 1 задания 1:)
plt.figure(figsize=(12, 7))
grid = np.mgrid[3.5:8.6:0.01, 1:5:0.01]
pos = np.empty(grid[0].shape + (2,))
pos[:, :, 0] = grid[0];
pos[:, :, 1] = grid[1]

# Вычисление плотности
gauss1 = sps.multivariate_normal(means[0][:2], covs[0][:2, :2])
gauss2 = sps.multivariate_normal(means[1][:2], covs[1][:2, :2])
gauss3 = sps.multivariate_normal(means[2][:2], covs[2][:2, :2])
density = (gauss1.pdf(pos) + gauss2.pdf(pos) + gauss3.pdf(pos)) / 3

plt.pcolormesh(grid[0], grid[1], density, cmap='Oranges')
plt.scatter(sample[:, 0], sample[:, 1], alpha=0.2)
CS = plt.contour(grid[0], grid[1], density, [0.01, 0.1, 0.2, 0.3, 0.4, 0.5])
plt.clabel(CS, fontsize=14, inline=1, fmt='%1.2f', cmap='Set3')
plt.show()
```



Вычислите условное математическое ожидание $\mathrm{E}(X|I\{T \neq k\} = 1)$ для всех $k = 1, 2, 3$, где X --- случайный вектор, имеющий распределение смеси. Постройте графики условной плотности $p_{X|I\{T \neq k\}}(x|1)$ в проекции на первые две координаты. Подберите хорошие значения линий уровня.

In [113]:

```
grid = np.mgrid[3.8:8.3:0.01, 1.4:4.8:0.01]
pos = np.empty(grid[0].shape + (2,))
pos[:, :, 0] = grid[0]
pos[:, :, 1] = grid[1]

density1 = (gauss2.pdf(pos) + gauss3.pdf(pos)) / 2
density2 = (gauss1.pdf(pos) + gauss3.pdf(pos)) / 2
density3 = (gauss1.pdf(pos) + gauss2.pdf(pos)) / 2

plt.figure(figsize=(15, 18))
for i, dens in enumerate([density1, density2, density_3]):
    plt.subplot(3, 1, i + 1)
    plt.pcolormesh(grid[0], grid[1], dens, cmap='Oranges')
    CS = plt.contour(grid[0], grid[1], dens,[0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.8])
    plt.clabel(CS, fontsize=14, inline=1, fmt='%1.2f', cmap='Set3')
plt.show()
```



Классифицируйте все пространство по принципу $k = \arg\max_k p_{X|I\{T=k\}}(x|1)$. Посчитайте долю ошибок на выборке. Нарисуйте классификацию всего пространства в проекции на пары координат (0, 1), (1, 3) и (2, 3), где закрасьте разными цветами области, которые образовались в результате классификации.

In [118]:

```
def argmax_k(x):
    return np.argmax(np.array([sps.multivariate_normal(mean=means[0], cov=covs[0]).pdf(x),
                                           sps.multivariate_normal(mean=means[1], cov=cov
                                           sps.multivariate_normal(mean=means[2], cov=cov
```

In [119]:

```
from sklearn.metrics import accuracy_score
accuracy_score(np.array(list(map(lambda x: argmax_k(x), sample))), classes)
```

Out[119]:

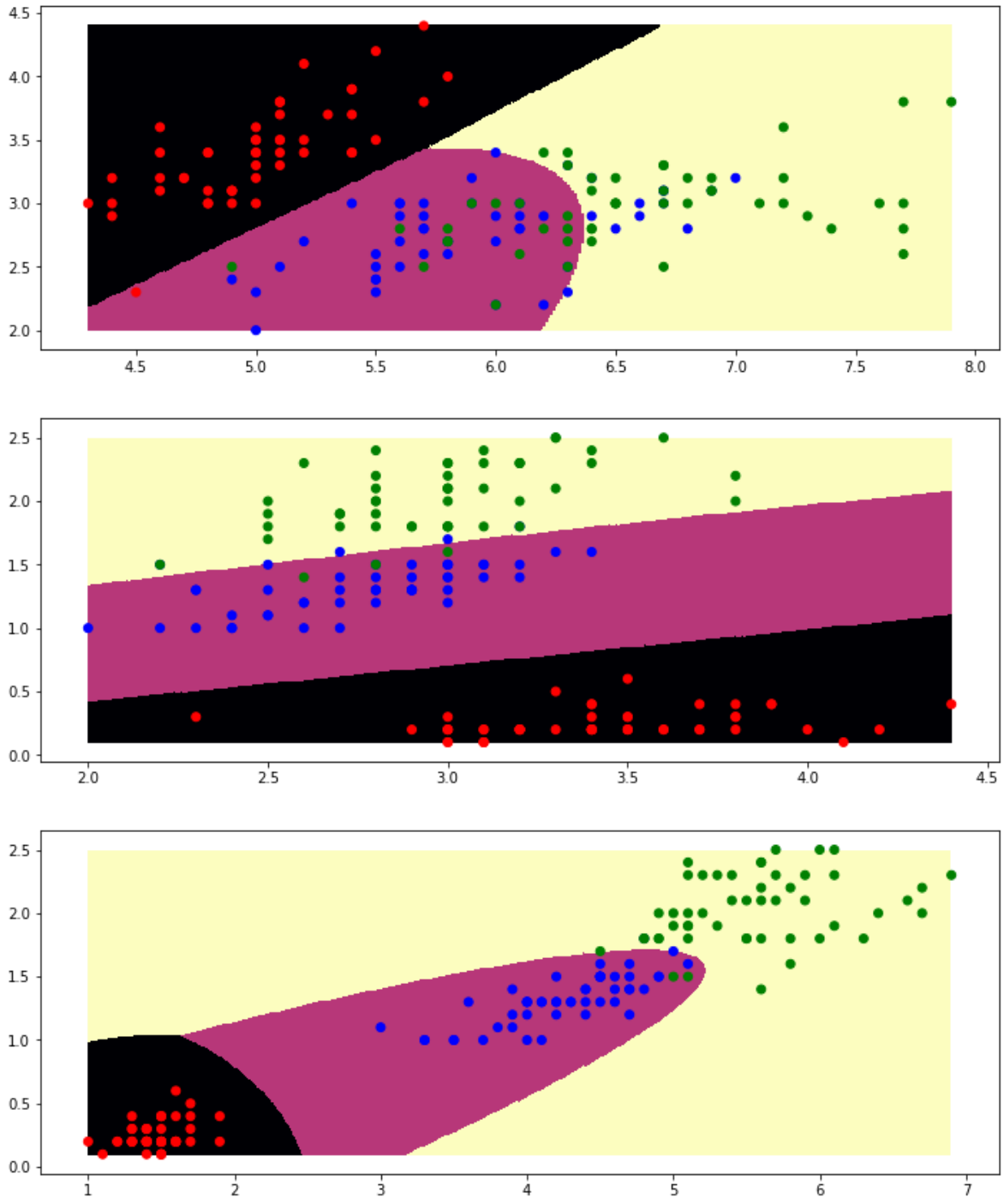
```
0.97999999999999998
```

In [120]:

```
def get_image(grid, dens1, dens2, dens3):
    pos = np.empty(grid[0].shape + (2,))
    pos[:, :, 0] = grid[0]; pos[:, :, 1] = grid[1]
    colors = np.array(list(map(lambda x:  np.argmax(x),
                               np.hstack((dens1.pdf(pos).reshape(-1, 1),
                                          dens2.pdf(pos).reshape(-1, 1),
                                          dens3.pdf(pos).reshape(-1, 1))
                                         ))
                           )
                      )
    return colors.reshape(grid[0].shape)
```

In [146]:

```
def proect(mean, cov, i1, i2):
    new_mean = mean[[i1, i2]]
    new_cov = np.array([[cov[i1][i1], cov[i1][i2]],
                        [cov[i2][i1], cov[i2][i2]]])
    return (new_mean, new_cov)

plt.figure(figsize=(12, 15))
for i, I in enumerate([(0, 1), (1, 3), (2, 3)]):
    grid = np.mgrid[np.min(sample[:, I[0]]):np.max(sample[:, I[0]]):1e-2,
                    np.min(sample[:, I[1]]):np.max(sample[:, I[1]]):1e-2]
    colors = get_image(grid, sps.multivariate_normal(*proect(means[0], covs[0], *I)),
              sps.multivariate_normal(*proect(means[1], covs[1], *I)),
              sps.multivariate_normal(*proect(means[2], covs[2], *I)))
    plt.subplot(3, 1, i + 1)
    plt.pcolormesh(grid[0], grid[1], colors, cmap='magma')
    colors_map = ['red', 'blue', 'green']
    plt.scatter(sample[:, I[0]], sample[:, I[1]],
                color=list(map(lambda x: colors_map[x], classes)))
plt.show()
```



Вывод:

Классификация по данному в условию признаку хорошо работает:)

Задача 5*. В предыдущей задача информация о принадлежности наблюдения конкретной компоненте смеси была известна заранее. Как выть в случае, если такой информации нет? Задача оценки параметров распределения смеси может быть решена с помощью иттерационного EM-алгоритма.

Опишите, как работает EM-алгоритм (это обязательное условие, при котором эта задача будет проверяться). Затем примените EM-алгоритм к Ирисам Фишера и к некоторым искусственно сгенерированным датасетам. Исследуйте, как результат зависит от параметров алгоритма. Сделайте вывод.

Разобраться в EM-алгоритме помогут: